AD-A042 125 MARYLAND UNIV COLLEGE PARK COMPUTER SCIENCE CENTER F/G 14/5
EXTRACTION OF TOPOLOGICAL INFORMATION FROM DIGITAL IMAGES,(U)
JUN 77 A ROSENFELD AF-AFOSR-3271-77
UNCLASSIFIED TR-547 AFOSR-TR-77-0806 NL

| OF |

AD
A042125

END
DATE
FILMED
8-77

(12)

AFOSR-TR- 77-0806

COMPUTER SCIENCE TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

(See 1473)

TR-547 June 1977
AFOSR-77-3271

EXTRACTION OF TOPOLOGICAL INFORMATION FROM DIGITAL IMAGES

Azriel Rosenfeld
Computer Science Center
University of Maryland
College Park, MD 20742

ABSTRACT

Geographic information is commonly derived from remote sensor imagery. The images are digitized and segmented into categories of interest, such as terrain or land use types. The regions resulting from this segmentation process can then be individually labelled, using connected component analysis or refinements of it. Polygonal boundary chains can be constructed for each of these regions, and the topological relationships among the regions can also be extracted.

This paper describes efficient algorithms for image segmentation into regions, region labelling, and construction of polygonal representations of regions. Such algorithms can serve as an interface between digital image data bases and polygon-based geographic information systems.

The support of the Directorate of Mathematical and Information Sciences, U. S. Air Force Office of Scientific Research, under Grant AFOSR-77-3271, is gratefully acknowledged, as is the help of Mrs. Shelly Rowe in preparing this paper. This paper is to be presented at the Advanced Study Symposium on Topological Data Structures for Geographic Information Systems, Dedham, MA, October 1977.

Azriel Rosenfeld
Computer Science Center
University of Maryland
College Park, Maryland 20742
UNITED STATES
301/454-4527

EXTRACTION OF TOPOLOGICAL INFORMATION FROM DIGITAL IMAGES

1. INTRODUCTION

This paper reviews techniques for extracting regions and region descriptions from digital images. A digital image is an array or matrix whose elements represent the reflectivities of points in the terrain (or in whatever type of scene gave rise to the image). In the case of multispectral imagery, each "multi-image" element is a vector whose components represent reflectivities of the same point in various spectral bands. To extract region-based information from an image, the following steps are generally necessary:

a) Segmentation of the image into subsets having different characteristics.

b) Decomposition of each subset into connected components, and determination of the topological relations (adjacency, surroundedness) between the components.

c) Representation of the individual components, e.g., by boundary chains or other types of polygons.

With each region (= connected component) can then be associated various types of descriptive information, relating to its spectral and textural content, to its size and shape, and so on.

The following sections of this paper will deal, respectively, with algorithms for image segmentation, connected component decomposition, and boundary representation of the components. We will use the following notation and terminology: The elements of an image will be called _points_ or _pixels_; a pixel will be identified by its position (x,y). The value of a pixel will be called its _gray level_, and will be denoted by f(x,y). For a pixel in a multi-image, we have a k-tuple of values $(f_1(x,y),\ldots, f_k(x,y))$, called the _spectral signature_ of the pixel at (x,y).

2. SEGMENTATION

When we segment an image into subsets of different types, we are classifying the pixels of the image into different classes. The basis for classifying a given point is (in general) provided by a collection of property values that we measure at the point; the subsets are characterized by particular combinations of these values. A brief review of basic segmentation methods will now be given; further information can be found in Chapter 8 of [3].

2.1 Gray Level Thresholding

If an image consists of light regions on a dark background or vice versa (e.g., clouds against sea surface, or water against land), we can classify its pixels into "light" and "dark" classes by simple thresholding. In other words, the gray level $f(x,y)$ at each point (x,y) is compared to a threshold t, and is classified according to whether $f(x,y) \geq t$ or $< t$.

If we know the statistics of the light and dark classes (e.g., that they have Gaussian distributions with given means and standard deviations), then we can determine mathematically where to put the threshold t in order to discriminate the classes with minimum error. If the class statistics are not known, we can decide where to put t by analyzing the clustering behavior of the image gray levels. In particular, suppose that we plot how many times each gray level occurs in the image; this plot is called the gray level histogram of the image. If this histogram has two peaks separated by a valley, then the peaks represent the light and dark regions, and they can be best distinguished by putting the threshold t at the gray level corresponding to the bottom of the valley.

2.2 Spectral Classification

Just as we can segment an ordinary image into sets of light and dark pixels based on the single property of gray level at each pixel, so we can segment a multi-image into subsets having given spectral characteristics, using the spectral signature of each pixel as a collection of property values. This is commonly done in analyzing multispectral remote sensor imagery; the subsets or classes may correspond to land use types, crop types, and so on.

We can think of the spectral signature $(f_1(x,y),\ldots,f_k(x,y))$ of the pixel at (x,y) as defining a point in a k-dimensional space. Thus to classify the pixel, we can divide the space into regions, such that the class to which a pixel is assigned is determined by the region into which its k-dimensional point falls. For example, a k-dimensional hyperplane, that divides the space into two half-spaces, is defined by an equation of the form

$$a_1z_1+\cdots+a_kz_k = t$$

The pixel whose signature is $(f_1(x,y),\ldots,f_k(x,y))$ falls into one or the other of these half-spaces according to whether

$$a_1f_1(x,y)+\cdots+a_kf_k(x,y) \geq t \text{ or } < t$$

Here again, if we know the statistics of the spectral signatures for the pixels belonging to each class, e.g., that they have multivariate Guassian distributions with given mean vectors and covariance matrices, we can in principle determine mathematically how the space should be partitioned in order to minimize the classification error. If the statistics are not known, we must decide how to partition the space by analyzing the clustering behavior of the pixel spectral signatures. For example, if these signatures form two compact, well-separated clusters in the space, we can conclude that there are two natural classes of pixels present in the image, and we can discriminate these classes by partitioning the space, say with a hyperplane, so that each cluster lies in a different part of the space. Evidently, this process is exactly analogous to that of gray level threshold selection; a threshold is simply a point that partitions a one-dimensional space (defined by the simple property of gray level) into two half-spaces.

2.3 Texture Discrimination

In order to segment an image into differently textured regions, it is necessary to make use of pixel properties whose values depend not only on the gray level of the pixel itself, but also on the gray levels of nearby pixels. We can illustrate this by the following simple examples.

Suppose that the image conatains two types of regions composed of small dark elements on a light background, but that in one type of region the elements are densely packed, while in the other type they are sparsely scattered. We cannot distinguish these types of regions by thresholding, since this can only discriminate the elements from the background, but cannot respond to differences in element density. However, we can make the regions discriminable by the simple device of blurring the image, i.e., replacing the gray level of each point by the average gray level computed over some neighborhood of that point. (We ignore here what happens at the edges of the image.) Under this process, the densely packed regions become generally dark, and the sparse regions light, and it is then possible to discriminate the regions by thresholding, as in Section 2.1.

As another example, suppose that one type of region is "busier" than the other -- e.g., that one type contains small, densely packed dark elements, while the other contains larger, less densely packed elements. Thus in the first type of region, transitions from light to dark are very frequent, while in the second type, they are rarer. Here again, we cannot discriminate the regions by thresholding, since this only distinguishes the elements from the background; nor can we be sure of doing it by averaging and thresholding, since the average gray level in both types of regions may be essentially the same. However, we can distinguish the regions by noting that the first type contains many more light/dark adjacencies than the second. Thus we can compute a measure of gray level difference at each point of the image (see Section 2.4), obtaining a new array in which high values are more frequent in the first type of region than in the second. This new array can then be segmented by averaging and thresholding, as in the preceding paragraph.

These two examples are representative of a large class of cases in which two types of regions differ with respect to the average value of some local property. (In the first case, the property is simply gray level; in the second case, it is gray level difference.) In such cases, the regions can be discriminated, in principle, by a three-step process of (a) computing the local property at each point; (b) averaging the results over a neighborhood of each point; and (c) thresholding the resulting array of average local property values. A wide variety of textural differences among regions can be discriminated in this way. Note that in all these cases we are still classifying image points on the basis of property measurements, though the property value at a point now depends on the gray levels in a neighborhood of the point.

2.4 Edge Detection and Template Matching

Segmentation need not involve discriminating among different types of regions in an image; it may instead involve detecting specific types of local patterns in the image. For example, one may want to distinguish edge points (i.e., points where the rate of change of gray level is high) from non-edge points, or points where some template closely matches the image from points where it does not match. Here again, we are classifying points based on the value of a local property -- the value of the gray level difference, or the degree of match with the template; in fact, we are thresholding the difference or match value in order to decide whether or not an edge or pattern is present.

Local pattern detection is important in a variety of image analysis applications. Examples are the detection of region boundaries (e.g., coastlines) or curve-like features (roads, rivers, lineaments, etc.) on remote sensor imagery; this is done using local patterns corresponding to short line segments having a range of orientations. A variety of match measures can be used for local pattern detection; the details will not be discussed here. Similarly, a variety of difference operators can be used for edge detection.

2.5 Sequential Methods of Segmentation

In all of the segmentation techniques discussed up to now, the image pixels are classified independently of one another; in other words, the decision made about one point does not depend on decisions about other points -- though it may depend on the gray levels of such points. These methods are often called "parallel", because the classification process could, if desired, be carried out for all pixels simultaneously.

Another class of segmentation methods are known as sequential, because they do take advantage of previous decisions when classifying each pixel. A standard example of this approach is <u>curve tracking</u>, in which, once a curve is detected at a point, we examine the neighbors of that point, and decide which of these neighbors continues the curve; then repeat the process for these neighbors; and so on. Here we are still classifying points based on property values (e.g., degree of match with a line-segment template); but the choice of property (e.g., slope of the template) and decision criterion (degree of match required for acceptance of a point) can now depend on results of previous decisions (e.g., in which direction the curve was headed, and how high its contrast was). The sequence in which points are examined also depends, of course, on the results found at previously examined points.

A more general example of sequential segmentation is <u>region growing</u>. Here we start with a single pixel, or with a uniform piece of the image (such as a connected component of constant gray level; see Section 3), and "grow" a homogeneous region by successively adding pieces whose addition does not violate a homogeneity criterion. Thus we are basically clustering pixels (or collections of them) based on similarity of property values; the decisions depend, at least to some degree, on the sequence in which the clusters are grown.

Sequential methods of segmentation are potentially more powerful than parallel methods, since they can take advantage of information obtained at previous steps in making subsequent decisions. However, they have a potential disadvantage with respect to computational cost, because they require the points of the image to be examined in an arbitrary sequence, or repeatedly. Parallel methods, on the other hand, can be implemented in a single, systematic scan through the image, since they can be applied in any desired order. We have emphasized simple parallel methods in this brief review, because they are both more generally applicable and less computationally costly. Of course, simple methods such as those described here will often not yield perfect segmentations, but they do provide a starting point for the development of more specialized methods.

3. CONNECTED COMPONENT EXTRACTION

Once an image has been segmented into subsets, we can further analyze the toplogical properties of these subsets. In particular, we can break them up into connected components, and determine the adjacency and surroundness relationships that hold among these components. Methods for carrying out these steps will now be reviewed; for further details see Chapter 9 of [3], particularly Section 9.1.

3.1 Connected Components

In a digital image, each point (x,y) has four horizontal and vertical neighbors

$$(x\pm1,y),\ (x,y\pm1)$$

and four diagonal neighbors

$$(x\pm1,y\pm1),\ (x\pm1,y\mp1)$$

(We ignore the special case of a point on the edge of the image.) For any points P,Q, a path from P to Q is a sequence of points

$$P = P_0, P_1, \ldots, P_n = Q$$

such that P_i is a neighbor of P_{i-1}, $1 \le i \le n$. Here n is called the length of the path.

Let S be any subset of the image. We say that the points P,Q of S are connected in S if there exists a path from P to Q that consists entirely of points of S. For any $P \in S$,

$$S_P = \{Q \in S \mid P \text{ and } Q \text{ are connected in } S\}$$

is called a connected component of S. Readily, S is the union of its connected components, and they are pairwise disjoint -- in other words, they constitute a partition of S.

It is important to point out that there are two versions of the above definitions, depending on whether we allow as neighbors of a point only its horizontal and vertical neighbors, or also its diagonal neighbors. If we do the latter, then the set whose points are indicated by *'s below

```
 *
*
```

is connected; if not, this set consists of two connected components, each having a single point.

3.2 Component Labelling

A number of algorithms have been devised for giving distinctive labels to the distinct connected components of a given set S. The most efficient of these algorithms, which we will now describe, requires only two systematic scans through the image to label all the components. We assume here that the points of S have been specially marked; for simplicity, we will refer to these points as 1's, and to the points not in S as 0's.

We scan the image row by row. On the first row, we assign a distinct label to the 1's in each run of 1's. For example, if the row looks like

001100010111

and we are using the numbers 2,3,4,... as labels, then the result of labeling this row is

002200030444

On succeeding rows, we treat each run ρ of 1's as follows:

a) If ρ is adjacent to no runs of 1's on the preceding row (i.e., no 1 in ρ is a neighbor of any 1 on the preceding row), we give its points the next available label that has not yet been used.

b) If ρ is adjacent to one run of 1's, say ρ', on the preceding row, we give its points the same label that was assigned to the points of ρ'.

c) If ρ is adjacent to two or more runs of 1's on the preceding row, we give its points one of their labels (say the least one), and we also record the fact that these labels are all "equivalent", i.e., they have been assigned to a single connected component.

When this scan of the image is complete, each point of S has a label, and no points belonging to different connected components have the same label; but it is quite possible that points belonging to the same connected component have different labels, whose equivalence was not discovered until later in the scan. To eliminate these superfluous labels, we process the list of recorded equivalences, and determine, for each label, the smallest label (say) that is equivalent to it. We can now scan the image again, and replace each label by its smallest equivalent label. After this second scan is finished, the points of each connected component all have the same label.

Variations of this algorithm exist in which we need examine only very small neighborhoods of each point of S, rather than examining entire runs of points of S. For example, let the points to the left of and above point P be

```
 B
CP
```

where P is a point of S. Suppose, for concreteness, that we allow only horizontal and vertical neighbors. Then we can assign a label to P according to the following rules:

a) If B = C = 0, give P the next available new label.
b) If B = 1 or C = 1, but not both, give P the same label as B or C.
c) If B = C = 1, give P the same label as C (say), and record the fact (if not already recorded) that the labels of B and C are equivalent.

This algorithm requires very little processing for each point P; but during the first scan, it will use many more labels than the first algorithm. For example, in cases like

```
  **
 **
**
```

the first * on each row will get a new label, since B = C = 0 for these *'s.

3.3 Component Properties and "Noise Cleaning"

The component labelling algorithm can also provide, as byproducts, a count of the number of components, and a measure of the area of (i.e., the number of points in) each component. In fact, the number of components is just the number of inequivalent labels that are used, and the area of a component is just the number of times that its label is used. Other properties of the components can also readily be computed, such as their perimeters, heights and widths, or their moments of various degrees, by keeping incremental count of various quantities as the components are being labeled. The details of these refinements will not be described here.

If the image has been poorly segmented, there may be many small "noise" components in the set S, or there may exist narrow gaps that break up components. Noise components can be eliminated using an area criterion -- i.e., discarding any component whose area is smaller than some threshold. Alternatively, they can be eliminated by a "shrinking and reexpanding" process that operates as follows: Change all 1's to 0's if they have 0's as neighbors; then change all 0's to 1's if they have 1's as neighbors. This process first shrinks S by erasing its border points (= points that have neighbors not in S), then expands S again by adding a border to it. However, parts of S that are nowhere more than two points wide will disappear completely under the shrinking, and so cannot be restored by the expansion. Note that this process eliminates thin, elongated parts of S even if they are very long; thus it should be used only if such parts are regarded as noise. The shrinking and reexpanding can be done in two scans of the image; or they can be done in a single scan, if larger neighborhoods of each point are examined (the details will not be given here).

Gaps in S can be mended by an expanding and reshrinking process exactly analogous to the shrinking and reexpanding process just described. Note that this will fill small holes and thin concavities, as well as briding thin gaps. These processes can be generalized to allow repeated shrinking and/or expanding, in order to eliminate noise components or gaps that have widths greater than 2.

3.4 Adjacency

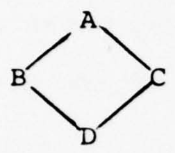
Let S and T be disjoint subsets of an image. We say that S and T are adjacent if some point of S is a neighbor of some point of T. If an image has been partitioned into labelled subsets, it is a straightforward matter to determine which pairs of these subsets are adjacent by scanning the image and examining the neighborhood of each labelled point. These adjacency relationships define a connected graph G whose nodes are the subsets and whose arcs are adjacencies; we call G the adjacency graph of the given partition.

An important special case of the adjacency graph arises when the partition consists of the connected components of a given set S and of its complement $\bar{S}$. In this case, it can be shown that G is a tree. It is straightforward to determine the adjacency relationships in the process of scanning the image to label the connected components of S and of $\bar{S}$ (both of which can be labelled simultaneously). Algorithms have also been devised for constructing the tree G explicitly (e.g., encoded as a string of parentheses) in a single scan of the image; the details will not be given here.

It should be pointed out that the conclusions in the preceding paragraph are valid only if we use opposite types of connectedness for S and for $\bar{S}$ -- i.e., if we allow diagonal neighbors for one of them, but not for the other. For example, if we do not allow diagonal neighbors for either, then in the image

$$\begin{array}{c}AB\\CD\end{array} \equiv \begin{array}{c}01\\10\end{array}$$

we have four connected components, two in S and two in $\overline{S}$, and the adjacency graph is



which is not a tree. Many other topological algorithms require the same assumption of opposite definitions for S and $\overline{S}$ -- for example, the border following algorithm to be given in Section 4.1.

The adjacency relationship considered in this section is a very simple one; it does not take into account the order in which various other sets touch S around its border, nor the fact that a given set may touch S several times. More will be said on this topic when we discuss region borders in Section 4.1.

3.5 Surroundness

Let S and T be disjoint subsets of an image. We say that T surrounds S if any path from a point of S to the edge of the picture must pass through T. (In other words: For all paths $P = P_0, P_1, \ldots, P_n = Q$ such that P is in S and Q is on the edge of the picture, some P_i must be in T,

$1 \le i \le n$.)

For a given pair of sets S, T, one can determine whether or not T surrounds S by constructing a border curve C of T (see Section 4) and testing whether or not some point of S is inside C. (The many methods of determining whether or not a point is inside a polygon will not be reviewed here.) For an arbitrary partition of an image into sets, it is tedious to determine all the surroundedness relations, since many pairs of sets must be examined.

The problem of determining all the surroundedness relations is greatly simplified in the case where the partition consists of the connected components of S and of $\overline{S}$. It can be shown that whenever a component U of S and a component V of $\overline{S}$ are adjacent, one of them must surround the other, unless they both touch the edge of the image. (Here again we must use opposite types of connectedness for S and $\overline{S}$; otherwise, the interior squares of a checkerboard all touch each other, but do not surround each other, and they are all components if we do not allow diagonal neighbors in either S or $\overline{S}$.) Let us construct the adjacency tree G for the components of S and $\overline{S}$, as in Section 2.4, and mark all the nodes of G corresponding to components that touch the edge of the image. For any two components U and V, let u and v be the corresponding nodes of G. Since in any tree there is only one path between any two given nodes, it is easy to construct the paths from u to each of the marked nodes. If v is on all of these paths, then V surrounds U, and conversely. Thus using G greatly simplifies determining all the surroundedness relations that hold between components of S and of $\overline{S}$.

4. COMPONENT REPRESENTATION

Let U be a connected set of points in a digital image, so that U consists of a single connected component. Although it is connected, U can still have a very complicated shape, and it may also have many holes -- i.e., there may be many components of $\overline{U}$ (the complement of U) that U surrounds. Thus U does not necessarily have a simple description.

The standard approach to describing a connected set U is to specify its border (the points where it is adjacent to $\overline{U}$, or to the edge of the image) by a set of curves. Note that many curves may be required, since U may have many holes. In this section we describe algorithms for tracking or "following" the borders of U (along which it meets various components of $\overline{U}$) and constructing these curves. For simplicity, we will assume that U does not touch the edge of the image. Further details on the material in this section can be found in Chapter 9 of [3].

The digital curve that represents a given border of U is a polygon, defined by sequences of "unit vectors" corresponding to the steps from point to point around the border. One can also approximate a (digital) curve by a polygon, using some error criterion to define how closely the sides of the polygon must fit the curve. Polygonal approximation techniques will not be considered further here.

A set U can also be represented in other ways, not involving its borders. For example, U can be represented by a "skeleton" consisting of the centers and radii of the maximal squares (say) that are centered at the points of U and contained in U. This representation is usually not as economical as one based on borders, but it has computational advantages in certain cases (e.g., it facilitates constructing unions and intersections of sets). Additional information about skeletons can be found in Chapter 9 of [3]; they will not be considered further here.

4.1 Border Following

Let U and V be subsets of a digital image. The V-border of U, denoted U_V, is the set of points of U that have points of V as neighbors. Even if U and V are both connected, U_V need not be connected, since V may touch U in many places.

On the other hand, let U be connected, and let V be a connected component of $\overline{U}$ that is adjacent to U; then U_V is a "closed curve" (we will not define this term precisely here). The same is true if we take S to be any set and let U be a component of S and V component of $\overline{S}$ that is adjacent to U. Once again, in these definitions, we must use opposite types of connectedness for U and V (i.e., for S and $\overline{S}$). We shall refer to the points of U as 1's and to the points of V as 0's.

We now describe an algorithm, called BF, for visiting all the points of U_V in sequence. Suppose, for concreteness, that we allow diagonal neighbors for U but not for V. BF assumes that we are initially given a pair of horizontally or vertically neighboring points u,v with $u = 1$ and $v = 0$. It first checks that u has some neighbor in U; if not, $U = \{u\}$ is a single point, and there is no border to follow. Otherwise, BF proceeds to construct a sequence of such point pairs $(u,v) = (u_0,v_0),\ldots,(u_n,v_n)$ by operating as follows:

a) Let the neighbors (horizontal, vertical, and diagonal) of u_i, in (say) clockwise order starting with v_i, be $w_1,w_2,\ldots,w_8$.

b) Let w_j be the first of the w's that lies in U. (There must be one, since as we shall see, u_{i-1} is a neighbor of u_i; if $i = 0$, we have already ruled out the case where u_0 has no neighbor in U).

c) Take w_j as u_{i+1} and w_{j-1} as v_{i+1}.

d) Repeat steps (a-c) until u_0 is found again (say as u_m), provided that v_0 is one of the w's that are examined when applying step (b) to u_m.

If we use diagonal neighbors for V rather than U, we must modify BF as follows: At step (b), let w_j be the first of the w's that lies in U and is a horizontal or vertical neighbor of u_i. If $w_{j-1} = 0$, take w_j as u_{i+1} and w_{j-1} as v_{i+1}; if $w_{j-1} = 1$, take w_{j-1} as u_{i+1} and w_{j-2} as v_{i+1}.

Successive u's found by BF are connected to one another in the U sense, and successive v's are connected to one another in the V sense. Also, each pair (u_i,v_i) is a horizontally or vertically adjacent pair of points, so that the u's all belong to U_v, and the v's to V_u. The proof that BF always works is rather difficult, and will not be given here. BF cannot stop as soon as it encounters u_0, since it may have to pass through u_0 twice, if U is thin at u_0 (e.g., $\boxed{\begin{array}{c} v_0 \leftarrow \\ {*}{*}{*}u_0{*}{*}{*} \end{array}}$).

A number of variations on BF are possible. One of them follows the "cracks" between U and V; here each "crack" is defined by a pair of points (u,v) in U and V, respectively. This algorithm finds the same u's as BF, but it may stay at some of them for several successive steps, as it moves through a succession of neighboring v's (w's in BF that have value 0). The details of this and of other algorithms for border following will not be given here.

Algorithms like BF will follow the border between U and any given V. As pointed out earlier, if U has holes, it will have borders with several different V's. These borders may have points in common, or one of them may even be contained in another (e.g., if U is a simple closed curve). We can use BF to follow all the borders; but some care is necessary in marking the borders that have already been followed, to insure that borders not yet followed are not incorrectly excluded from further consideration. Incidentally, BF follows hole borders (= borders with V's such that U surrounds V) counterclockwise, and outer borders clockwise; this is a result of our arbitrary decision to use a clockwise ordering of the neighbors in step (a).

If the image has been segmented into several types of regions, and each of these has been labelled distinctively, then when we follow the borders of U, we can check at each border point P just which region(s) U is adjacent to at P. We can thus construct an oriented multigraph representing the adjacency sequences that occur around each border. The details of this process will not be considered further here.

4.2 Chain Coding and Object Reconstruction

Since successive u's found by BF are (horizontal, vertical, or diagonal) neighbors, we can describe the border traced by BF as a succession of moves from neighbor to neighbor. Let us identify the eight neighbors of a point (*) by numbers 0,1,...,7 as follows:

$$\begin{array}{ccc} 3 & 2 & 1 \\ 4 & * & 0 \\ 5 & 6 & 7 \end{array}$$

(mnemonic: The ith neighbor is the one in direction $45i°$, where angles are measured counterclockwise from the positive x-axis). Thus the succession of BF's moves can be represented by a sequence of 3-bit numbers. Such a sequence is called a chain code. A review of chain coding can be found in [1].

If we are given the chain codes of the borders of a set U, we can reconstruct U (as a set of 1's in a digital image), provided that we are also given, for each border U_V, an initial pair of points (u,v) with u in U and v in V. The reconstruction process is as follows: Mark the points u, v as 1, 0, respectively. The chain code defines which neighbor of $u = v_0$ is the next border point u_1. Taking the neighbors of u_0 in clockwise order starting with v_0, let v_1 be the neighbor immediately preceding v_1. Mark u_1, v_1 as 1,0, respectively; also mark all the neighbors of u_0 between v_0 and v_1 as 0. (The diagonal neighbors are marked only if we are using diagonal neighbors in U; otherwise they are left blank.) Repeat the process, with (u_1,v_1) in place of (u_0,v_0), and continue in this way until the chain is finished, which brings the border back to u_0. This process marks all the points of U_v as 1's, and all the points of V_u as 0's.

When all of the borders and "coborders" of U have been marked in this way, we can "fill in" the interior of U using a row-by-row scan of the image. The filling-in process is based on the observation that any interior point of U must belong to a horizontal run of interior points, having border points at its ends. Thus as we scan row by row, whenever we find a 1, we change blanks to 1's as we move to the right, but stop if we reach a 0. When the scan is complete, all of U will be marked with 1's. If desired, we can mark all of $\overline{U}$ with 0's in the same scan.

4.3 One-Pass Chain Coding

BF extracts borders one at a time, and it also requires access to the image in an arbitrary sequence, since the borders are arbitrary curves. In this section we describe an algorithm for simultaneously constructing the chain codes of all borders of all objects in the course of a single row-by-row scan of the image.

The algorithm operates as follows: On the top row of the image, it creates a chain code of the form 00...0 for each run of object points; these correspond to the tops of objects, followed clockwise. On any subsequent row, we already have a set of chains (possibly empty) that meet the preceding row; we assume that we know which end of which run corresponds to each end of each of these chains.

We now compare the positions of the runs on the current row (n) with those on the preceding row (n-1).

a) If ρ is a run on row n that is not adjacent to any run on row n-1, we initiate a new chain for ρ, just as we did for each run on the first row.

b) If ρ is a run on row n-1 that is not adjacent to any run on row n, we create a chain of the form 44...4, corresponding to the bottom of the run, and link it to the chain ends associated with the ends of the run (i.e., if these chains are κ and λ, at the right and left ends of ρ respectively, we join them to form $\kappa 44...4\lambda$). These chain ends have now been joined together and need no longer be tracked.

c) If run ρ on row n-1 is adjacent to rows $\rho_1,...,\rho_r$ on row n, we extend ρ's chain ends to join the left end of ρ with the left end of ρ_1, and the right end of ρ with the right end of ρ_r. For example, in the situation

```
    **********      ρ
****     **       ρ1,ρ2
```

if the chains associated with the left and right ends of ρ are κ and λ, we adjoin 001 to the beginning of κ, and 445 to the end of λ. Thus the beginning of the chain 001κ is now associated with the left end of ρ_2, and the end of the chain $\lambda445$ with the right end of ρ_2.

In addition, we create new chains corresponding to the border segments at the bottom of ρ between successive ρ_i's; these are all of the form 344...45. For example, in the case illustrated above, we create the chain 34445, and associate its beginning with the left end of ρ_2, and its end with the right end of ρ_1.

d) Similarly, if runs $\rho_1,\ldots,\rho_r$ on row n-1 are adjacent to run ρ on row n, we extend the chains corresponding to the left end of ρ_1 and the right end of ρ_r to join them with the left and right ends of ρ, respectively. For example, in the situation

```
****    **        ρ1,ρ2
   *********      ρ
```

if the chains associated withe the left end of ρ_1 and right end of ρ_2 are κ and λ, respectively, we adjoin 007 to the beginning of κ, and 700 to the end of λ. Thus the beginning of chain 007κ is now associated with the left end of ρ_1 and the end of chain $\lambda 700$ with its right end.

In addition, we join up the chains associated with the remaining ends of $\rho_1,\ldots,\rho_r$, using chain segments corresponding to the border segments at the top of ρ between the ρ_i's; these are all of the form 700...01. For example, in the case illustrated above, if μ,ν are the chains associated with the right end of ρ_1 and left end of ρ_2, respectively, we link them to form the chain $\mu 70001\nu$. The end of μ and beginning of ν have now been joined together and need no longer be tracked.

When the row by row scan of the image is complete, this process will have created chain codes of all the object borders in the image. Of course, we should also save the locations of a pair of adjacent object and background points at the start of each chain, so that the objects can be reconstructed from the chains. Note that a final border chain may be the result of linking together many different pieces, since the tracking of the border may have begun at many places; thus we may have saved many point pairs that turn out to be on the same border, and we can discard the superfluous ones.

As in Section 4.1, an alternative algorithm for one-scan border chain code construction can be devised that looks only at a small neighborhood of each object point, rather than at entire runs of object points. The details of such an algorithm will not be given here; they can be found in [2].

One could also devise an algorithm that creates an image of all the object borders in a single row-by-row scan, given the chain codes and the initial point pairs. This would require the chain codes to be stored in pieces, starting at each locally highest part of each border, and with an initial point pair for each such part. The use of such a storage scheme for chain codes should facilitate efficient conversion from chain code to digital image form and vice versa.

5. CONCLUDING REMARKS

This paper has described some basic algorithms for extracting object border descriptions from a digital image. The image is segmented into subsets, these are analyzed into connected components, and the closed-curve borders between pairs of the components are tracked, using either a border-following or row-by-row process. The reconversion from object borders to segmented digital image (array of 0's and 1's) was also discussed. Algorithms such as those described here play a central role in interfacing digital image data with polygonal region descriptions.

BIBLIOGRAPHY

1. H. Freeman, Computer processing of line drawing images, Computing Surveys 6, 1974, 57-97.

2. D. L. Milgram, Constructing trees for region description, Technical Report 541, Computer Science Center, Univ. of Maryland, College Park, MD, June 1977.

3. A. Rosenfeld and A. C. Kak, Digital Picture Processing, Academic Press, New York, 1976.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (19) (18) AFOSR-TR-77-0806	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) EXTRACTION OF TOPOLOGICAL INFORMATION FROM DIGITAL IMAGES,		5. TYPE OF REPORT & PERIOD COVERED Interim
		6. PERFORMING ORG. REPORT NUMBER (14) TR-547
7. AUTHOR(s) (10) Azriel Rosenfeld		8. CONTRACT OR GRANT NUMBER(s) (15) AF-AFOSR-~~77~~-3271-77
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Ctr. Univ. of Maryland College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F (16) 2304/A2 (17) A2
11. CONTROLLING OFFICE NAME AND ADDRESS Math.& Info. Sciences, AFOSR/NM Bolling AFB Wash., DC 20332		12. REPORT DATE (11) June 1977
		13. NUMBER OF PAGES 32 (12) 35p.
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Picture processing
Pattern recognition
Segmentation
Topological properties
Chain coding

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Geographic information is commonly derived from remote sensor imagery. The images are digitized and segmented into categories of interest, such as terrain or land use types. The regions resulting from this segmentation process can then be individually labelled, using connected component analysis or refinements of it. Polygonal boundary chains can be constructed for each of these regions, and the topological relationships among the regions can also be extracted.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE

403018 [illegible]

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 77-0806	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) EXTRACTION OF TOPOLOGICAL INFORMATION FROM DIGITAL IMAGES		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER TR-547
7. AUTHOR(s) Azriel Rosenfeld		8. CONTRACT OR GRANT NUMBER(s) AFOSR-77-3271
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Ctr. Univ. of Maryland College Park, MD 20742		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Math.& Info. Sciences, AFOSR/NM Bolling AFB Wash., DC 20332		12. REPORT DATE June 1977
		13. NUMBER OF PAGES 32
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Type page

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Picture processing	Topological properties
Pattern recognition	Chain coding
Segmentation	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Geographic information is commonly derived from remote sensor imagery. The images are digitized and segmented into categories of interest, such as terrain or land use types. The regions resulting from this segmentation process can then be individually labelled, using connected component analysis or refinements of it. Polygonal boundary chains can be constructed for each of these regions, and the topological relationships among the regions can also be extracted.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT, cont'd.

This paper describes efficient algorithms for image segmentation into regions, region labelling, and construction of polygonal representations of regions. Such algorithms can serve as an interface between digital image data bases and polygon-based geographic information systems.